UNIVERSITAT DE BARCELONA

FUNDAMENTALS OF DATA SCIENCE MASTER'S THESIS

Using Recurrent Neural Networks to predict the time for an event

Author:
Manel Maragall Cambra

Supervisor:
Jordi Vitrià

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamentals of Data Science

in the

Facultat de Matemàtiques i Informàtica

September 2, 2018

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Using Recurrent Neural Networks to predict the time for an event

by Manel Maragall Cambra

One of the main concerns of the manufacturing industry is the constant threat of unplanned stops. Even if the maintenance guidelines are followed for all the components of the line, these downtimes are common and they affect the productivity. Most of what is done nowadays in the manufacturing plants involves classic statistics, and sometimes online monitoring. However, in most of the industries the data related to the process is monitored and saved for regulatory purposes. Unfortunately it's barely used, while the actual technologies offer a wide horizon of possibilities.

The time to an event is a primary outcome of interest in many fields e.g., medical research, customer churn, etc. And we think that it's also very interesting for Predictive Maintenance. The time to an event (or in this context time to failure) is typically positively skewed, subject to censoring, and explained by time varying variables. Therefore conventional statistic learning techniques such as linear regression or random forests don't apply. Instead we have to relate on more complex methods.

In particular we focus on the WTTE-RNN framework proposed by Egil Martinsson, which employs Recurrent Neural Networks to predict the parameters of a Weibull Distribution. The result is a flexible and powerful model specially suited for time-distributed data that can be organized in batches.

Acknowledgements

First of all, I would like thank Professor Jordi Vitrià to find some time to be my project advisor. His remarks and suggestions have been of great importance to guide the course of this thesis. Also I would like to express my gratitude to Egil Martinsson, for not only having a great idea but also being patient and accessible to my constant questions.

I would also like to thank Llorenç Domingo and my other colleagues from Bigfinite for their support, and for offering me the flexibility I needed to attend the program during these last two years.

I'm also very grateful to my parents Cristina and Manel, they have always been there when I needed a boost. I know reading through this text has not been easy. Finally I'll always owe one to my partner Belén, she has made sure that I get ahead with the thesis and the whole MSc. Thank you very much!

Contents

Chapter 1

Introduction

There are a few concepts that are relevant to comprehend the work that has been done in this project. These are mostly statistic notions that will be addressed by complexity, aiming for a simple and schematic overview. Deepening into the theoretical basis of some of these concepts only when needed.

To understand the problem of predicting the time to an event, we have to consider the structuring of the data, or rather the shortcomings we can find in the observations. The next step is getting to know the suggested statistic distribution (Martinsson, 2016) to model this problem. Wrapping it with Deep Learning, an statistic learning framework that's getting a lot of attention these days.

Since we will exclusively focus on the branch of Recurrent Neural Networks (RNN), I suggest a chronological approach (Nielsen, 2015) for readers who are not familiar with Deep Learning. In Nielsen's blog, the basic units that conform a Multi-Layer Perceptron are introduced in an understandable way.

1.1 Censoring

Within statistics, *censoring* is defined as a condition in which the value of a measurement or observation is only partially known. This turns out to be a common phenomena in many fields, such as Health, Life Sciences, Engineering, etc. And of course, when studying the waiting time of an event, it's possible and even likely, that such event has not been observed from the start to the end.

Martinsson identifies *waiting times* T as an unbounded, always positive interval $T \in [0, \infty)$. And refers to the successive events occurring in the time-line as *recurrent events*. If we understand T as some positive random variable indicating the time to an event, we say a datapoint drawn from T is *censored* whenever we haven't observed it pointwise. Given an instant of time t, we can find the following situations:

- **Uncensored data**: The event has occurred exactly at time t. $T = t$
- **Left-censored data**: At time t we know the event has already occurred $T \in [0, t)$, but we don't know exactly when.
- **Right-censored data**: We know the event will occur after time t but we don't know exactly when. $T \in (t, \text{inf})$
- **Interval-censored data**: Given times t_1, t_2 where $t_1 <= t_2$, we know the event has occurred between t_1 and t_2. $T \in (t_1, t_2)$

1.2 The Weibull Distribution

The Weibull Distribution is a unimodal statistic distribution that got very popular in the 70's. Back then it was said to be "universal" and it even got some notorious disclaimers (Gorski, 1968). Having said that, it's a really expressive distribution that is still very used nowadays.

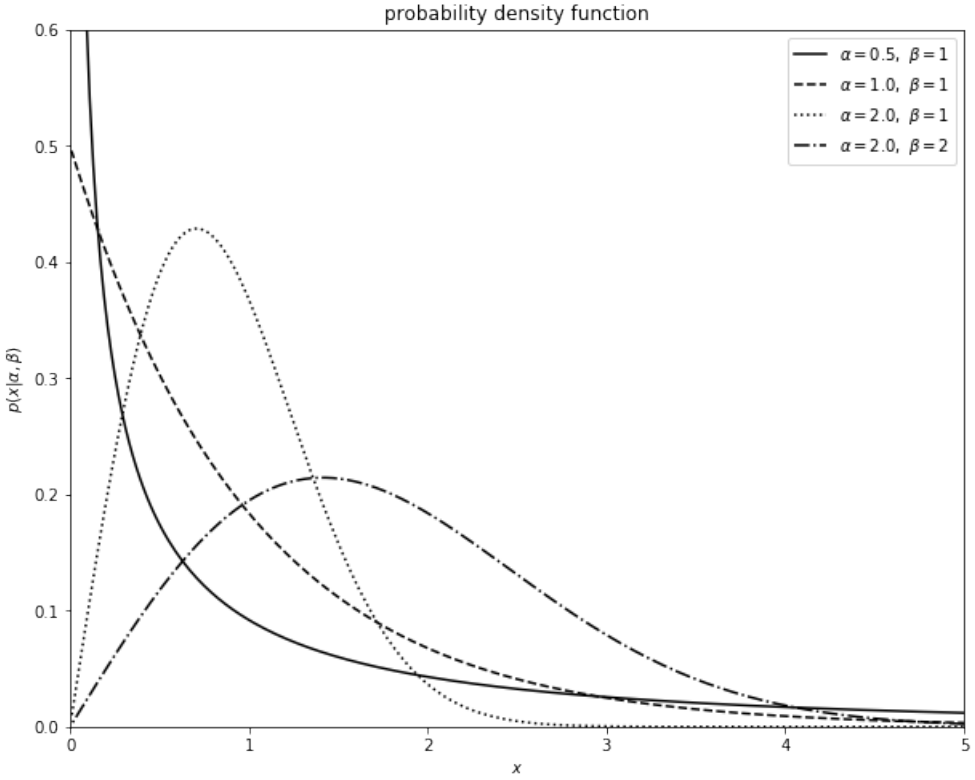


FIGURE 1.1: The Weibull Distribution pdf with 2 parameters α and β

Besides an infinite spike or an infinite flat probability density function f, the Weibull Distribution can relate to many other statistic distributions (Equation 1.1). When modelling the time to failure, it provides a distribution for which the failure rate is proportional to a power of time.

$$f(x) = \begin{cases} \left(\frac{\alpha}{\beta}\right)\left(\frac{x}{\beta}\right)^{\alpha-1} e^{-\left(\frac{x}{\beta}\right)^{\alpha}} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{1.1}$$

The shape parameter α[1] can be interpreted directly as follows; $\alpha = 1$ resembles the Exponential Distribution, indicating that the failure rate (Equation 1.4) is constant over time, whereas $\alpha > 1$ indicates that there is an "aging process", so the failure rate increases with the time. In particular, $\alpha = 2$ and $\beta = \sqrt{2\sigma}$ coincides with the Rayleigh Distribution. On the contrary, $\alpha < 1$ would indicate that the failure rate

[1] Apparently there are many variants in the notation of the parameters. In this paper we refer as α and β to the shape and scale of the 2-parameter Weibull Distribution.

decreases with the time.

$$F(x) = \begin{cases} 1 - e^{-\left(\frac{x}{\beta}\right)^{\alpha}} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{1.2}$$

As for the cumulative distribution function F, the Weibull Distribution presents a closed form that is also numerically stable (Equation 1.2).

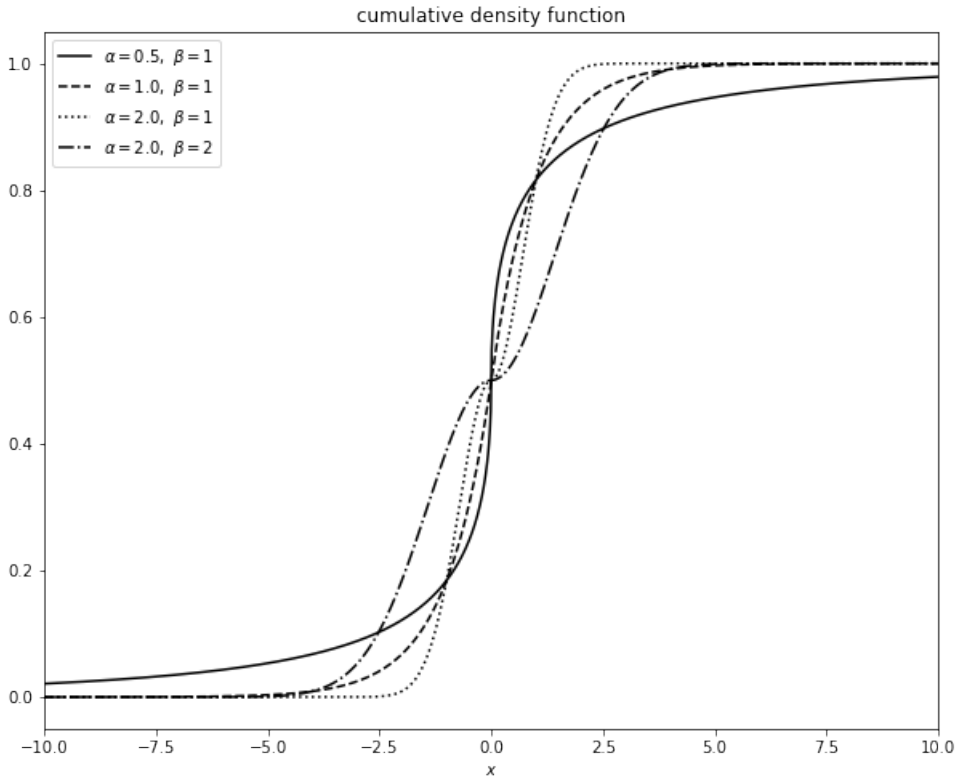


FIGURE 1.2: The Weibull Distribution pdf with 2 parameters α and β

The Weibull Distribution is commonly used in *Survival Analysis*, a branch of statistics that analyzes the expected duration of time until one or more events happen. In this discipline the object of primary interest is the survival function S.

$$S(t) = P(T > t) = 1 - F(t) = e^{-\left(\frac{t}{\beta}\right)^{\alpha}} \tag{1.3}$$

Where T is a random variable denoting the time to an event, t is some moment of time, and P denotes the probability. If we suppose that an item has survived for time t, and we want to know the probability that it will not survive for an additional time dt, we have the hazard function λ.

$$\lambda(t) = \lim_{dt->0} \frac{P(t \leq T \leq t + dt)}{dt \cdot S(t)} = \frac{f(t)}{S(t)} = \left(\frac{\alpha}{\beta}\right)\left(\frac{t}{\beta}\right)^{\alpha-1} \tag{1.4}$$

The hazard function is also known as *hazard rate*, *failure rate* in the field of reliability engineering, and *force of mortality* μ in demographics. An extension of this idea is the

accumulation of the hazard over time, a.k.a the cumulative hazard function Λ.

$$\Lambda(t) = \int_0^t \lambda(u)du = -\ln S(t) = \left(\frac{t}{\beta}\right)^{\alpha} \tag{1.5}$$

As it can be seen, we can express the probability density function f and the cumulative distribution function F of the Weibull Distribution through these concepts.

$$f(t) = \lambda(t) \cdot S(t) = \lambda(t) \cdot e^{-\Lambda(t)} \tag{1.6}$$

$$F(t) = 1 - S(t) = 1 - e^{-\Lambda(t)} \tag{1.7}$$

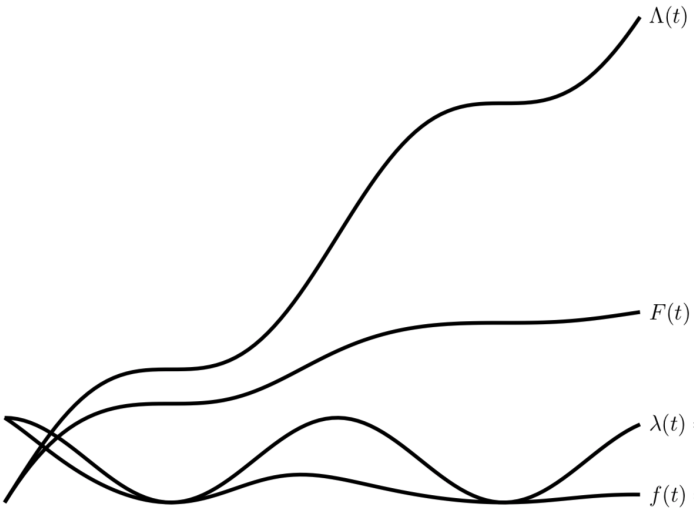


FIGURE 1.3: Relationship between Λ, λ, F and f from Martinsson, 2016

Additionally, the cdf of the Weibull Distribution is invertible[2] and therefore the quantile[3] has a closed form.

$$Q(p) = \beta(-\ln(1-p))^{\frac{1}{\alpha}} \tag{1.8}$$

Finally, these are the forms for the mean, variance, mode and median.

$$\mu = \beta\Gamma(1+\alpha^{-1}) \tag{1.9}$$

$$\sigma^2 = \beta^2[\Gamma(1+2\alpha^{-1}) - \Gamma^2(1+\alpha^{-1})] \tag{1.10}$$

$$mode = \beta\left(\frac{\alpha-1}{\alpha}\right)^{1/\alpha} \tag{1.11}$$

$$median = \beta\sqrt[\alpha]{\ln 2} \tag{1.12}$$

[2] An inverse function (or anti-function) is a function that "reverses" another function: if the function f applied to an input x gives a result of y, then applying its inverse function g to y gives the result x, and vice versa, i.e., $f(x) = y \iff g(y) = x$.

[3] The quantile can be used to sample the distribution by taking uniform samples of a number u, where $0 \leq u \leq 1$, interpreted as a probability p. This procedure is called Inverse Transform Sampling.

1.3 Recurrent Neural Networks

As you read this, you understand each word based on the previous words. You don't throw everything away and start thinking from scratch again. Your thoughts have **persistence**. So do Recurrent Neural Networks.

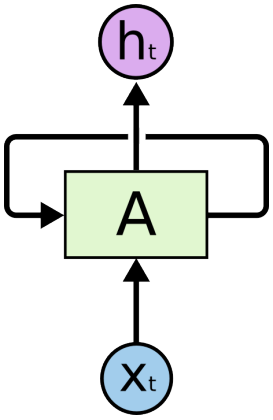


FIGURE 1.4: Rolled RNN

In simple terms, the differential property that RNNs incorporate, and traditional networks don't, are **loops**. In fact, these loops within them keep the short term memory flowing. Cristopher Olah (*Understanding LSTM Networks*) provides an excellent explanation of these ideas, the chapter is heavily borrowed from his blog and so are the figures (1.4, 1.5, 1.6).

1.3.1 Long Short Term Memory Networks

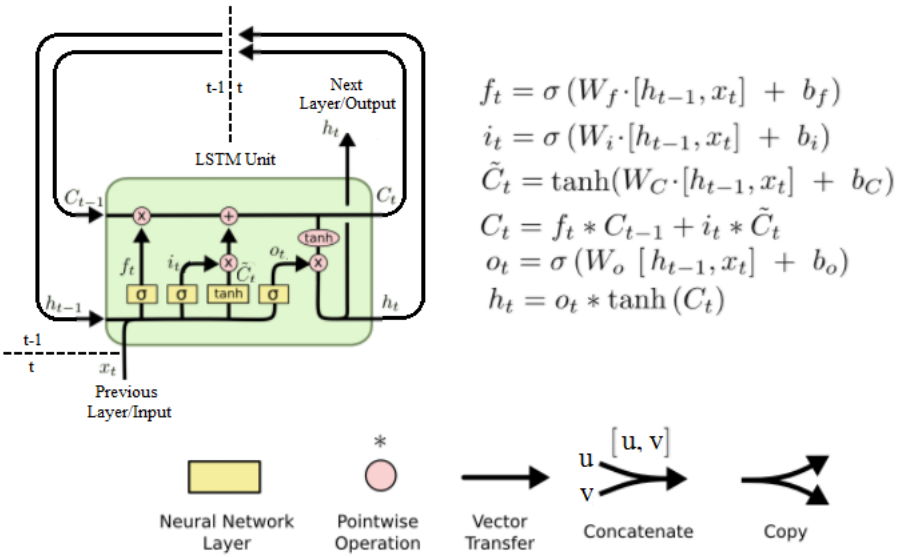
Usually called LSTMs, these are a special kind of RNN that are in theory more capable of learning long term dependencies. In this case, the mysterious loops that we introduced before are actually a chained repetition of these "modules", just like an standard RNN. But much more complex.

Each of the modules, sometimes referred as *cells*, looks something like Figure 1.5 and there is a lot going on inside. Again, you can find a detailed explanation in the blog. In any case, I will try to provide a basic overview; given a moment of time t, that represents the *th* event in the sequence. C_t is the horizontal line running through the top of the diagram. Along this line it's easy for the information to flow unchanged, but it can also be altered by the *gates*. The first gate from the left f_t is the forget layer, and decides how much of the information coming from the previous cell will persist. On the other hand, the product of $\tilde{C}_t$ and i_t takes care off the new information. Basically, these are new candidate values scaled by how much we want to update each state value. Thus:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{1.13}$$

The first part $f_t \cdot C_{t-1}$ of the Equation 1.13 gets rid off previous information, and $i_t \cdot \tilde{C}_t$ adds new values. Finally, we decide what are we going to output h_t, based on a filtered version of the cell state:

$$h_t = \sigma_t \cdot tanh(C_t) \tag{1.14}$$



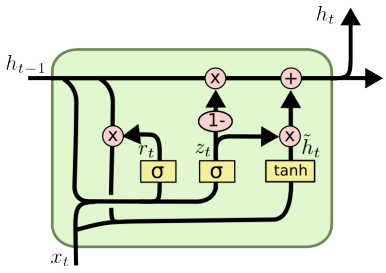
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

FIGURE 1.5: Inside a Long Short Term Memory module

Notice that the inner state C_t, the output mask σ_t and the output state h_t should have the **same** dimension. The amount of *units* of these layers depends on the complexity of the problem that we are trying to model. Therefore the selection of these units, along with the number of passed events to observe, are **very important hyper-parameters for the model**.

1.3.2 Gated Recurrent Unit

One of the variants of the LSTM cell is the Gated Recurrent Unit, usually referred as GRU (Cho et al., 2014). Cho proposed another twist in the magic happening inside the module, we can visualize it in Figure 1.6. It combines the forget and input gates, and it also merges the cell state and hidden state. The result is simpler and faster than standard LSTM models, and has been growing increasingly popular.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

FIGURE 1.6: Inside a Gated Recurrent Unit

Although LSTM and GRU cells have achieved outstanding results, it's reasonable to think that there exist alternative architectures that could obtain better results. In fact, some have ventured into finding new proposals (Jozefowicz, Zaremba, and Sutskever, 2015), by evaluating over 10.000 RNN architectures.

Chapter 2

Motivation and Goals

At my work place I have been involved with Manufacturing in the Pharmaceutical Industry for the past two years. It turns out that until recently, drug manufacturing was subject to very strict closed procedures. In simple terms, following these "recipes" can cause similar accidents to the ones that happen in domestic kitchens. If I may metaphor; after baking a cake, it's possible that we open the oven and it's not done. This can be due to *external factors*, such as atmospheric conditions, raw materials or unexpected issues with the yeasts. It turns out these issues are rather common when dealing with living things (e.g., bacteria).

Until recently, an issue in pharma meant throwing the cake to the trash. This can be measured in the order of millions of dollars in some cases. Fortunately, the Food and Drug Administration (FDA) and the other agencies are starting to allow "science" during the process. This also allows for a wide variety of mechanisms to provide real time information about the process, and more importantly, for *decisions* to be made.

2.1 The Pharmaceutical Industry

In pharma manufacturing, there are a bunch of notorious KPIs that are well known across the globe. Perhaps the more interesting one is the Overall Equipment Effectiveness (OEE), this metric takes into account the *quality*, *availability*, and *performance* to monitor the production of the manufacturing lines.

$$Quality = \frac{\sum GoodUnits}{TotalUnits(\sum GoodUnits + \sum Scraps)} \tag{2.1}$$

$$Availability = 1 - \frac{\sum NonPlannedStops}{PlannedProductionTime} \tag{2.2}$$

$$Performance = 1 - \frac{\sum SmallStops + \sum ReducedSpeedLoss}{PlannedProductionTime} \tag{2.3}$$

$$OEE = Quality \cdot (Availability + Performance - 100) \tag{2.4}$$

As we can see in the Equation 2.1, the OEE is greatly affected by the scraps, which are the rejected units. Scraps are typically related to unplanned stops in the line. Knowing in advance when there is going to be a failure (time to failure) can advert

the line operators so they prevent it, or at least prepare for a response. This of course may require more context, for instance what components are going to fail, or even what caused the failure. In any case, predicting the failures can be very interesting for this classical industry that is just starting to grasp the potential of Machine Learning.

2.2 Predictive Maintenance

In manufacturing, the techniques designed to help to determine the condition of in-service equipment, in order to predict when maintenance should be performed, are typically referred as Predictive Maintenance. The ultimate goal of this approach is to perform maintenance at a scheduled point in time when the maintenance activity is most cost-effective and before the equipment loses performance within a threshold.

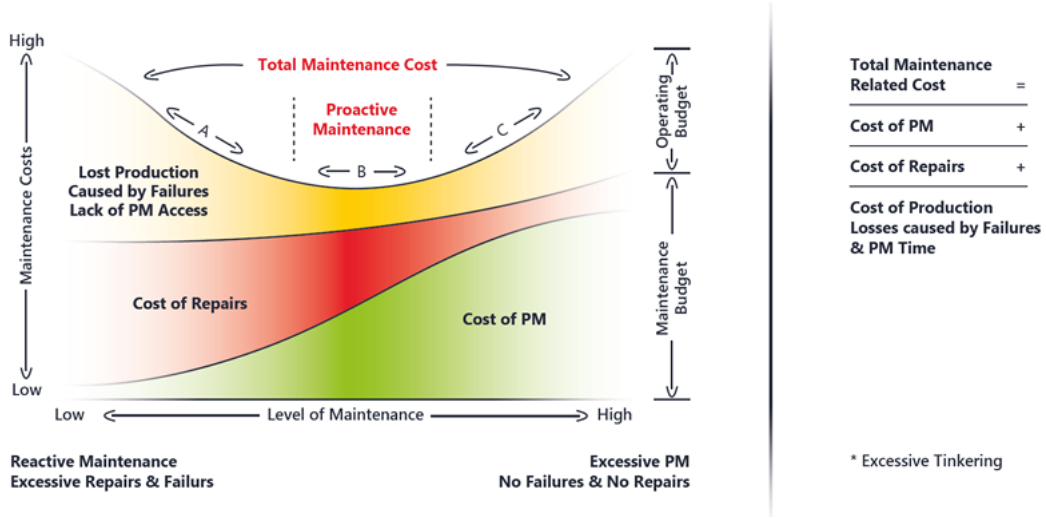


FIGURE 2.1: Maintenance vs cost from ureason

In the last decade, the availability of industrial internet of things (IIoT) devices has made it possible to monitor the machine continuously with wireless sensors, in order to assess the degradation of the components and predict the failures ahead of time. This new ways to monitor the equipment condition continuously (Online Monitoring), bring new lines of research to the manufacturing industries (Amruthnath and Gupta, 2018) that most likely will redefine how things are done.

The *goal* therefore is to model time to failure in order to solve the needs for Predictive Maintenance. These are the desirable properties of the model:

- **Performance**: Able to proof effectiveness in selected datasets.
- **Flexibility**: Should perform well in different processes (low variance).
- **Confidence**: Avoid costs for false positives. Somehow measure the uncertainty of the prediction.

Chapter 3

State of the Art

Time To Failure (TTF) has been tackled in many ways over the years. I present here a short selection of these techniques that approach the problem of Predictive Maintenance from **different perspectives**. We will see that the methods that take into consideration the possibility of having censored data (chapter 1.1), usually involve some cumbersome workarounds. Lastly we will focus on the Weibull Time to Event (Martinsson, 2016). It will be explained on chapter 3.4, but it basically approaches TTF in a very *natural*, *powerful* and *flexible* way.

3.1 Control Charts

Believe it or not most of what it's done in the field of Predictive Maintenance in manufacturing is related to Control Charts. First described by Walter A. Shewhart in the early ninenteen hundreds, this classical technique is based on routinely monitor *quality*. Where quality can be defined by one or multiple parameters.

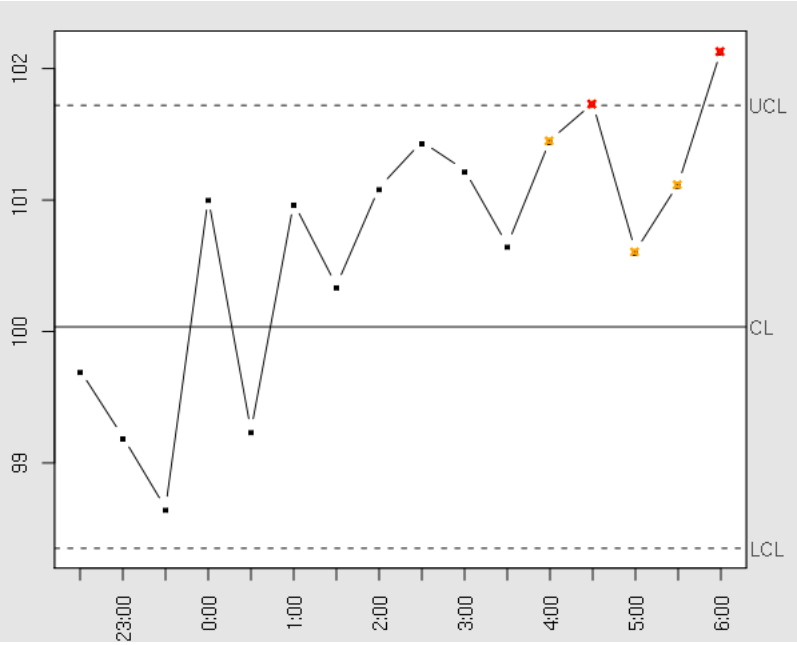


FIGURE 3.1: Control Chart from Wikipedia

The control chart (Figure 3.1) shows the value of the quality characteristic versus the number of produced units, or sometimes versus the time. In general, the chart contains a center line that represents the mean value for the in-control process, which

can be composed of multiple groups (typically substations of a manufacturing line). Two other horizontal lines, called the upper control limit UCL and the lower control limit LCL, are also shown on the chart. These control limits are chosen so that almost all of the data points will fall within these limits, therefore precision is preferred over recall.

In the U.S., **whether the data is normally distributed or not** (NIST/SEMATECH, 2003), it is an acceptable practice to base the control limits upon a multiple of the standard deviation. Usually this multiple is 3 and thus the limits are called 3σ limits. This term is used whether the standard deviation σ is the universe or population parameter, or some estimate thereof, or simply a "standard value" for control chart purposes. It's inferred from the context what standard deviation is involved. To sum it up, control charts monitor the process in real time but do not provide forward-looking insights about the quality.

3.2 Survival Models

In Survival Analysis the time to an event is usually referred as survival time, and it's studied to answer questions such as: How do particular circumstances affect the probability of survival? Which proportion of a population will survive past a certain time t? At which rate will the individuals of the population fail? To answer this questions we usually have to deal with time varying right censored data.

We will present the family of Proportional Hazard models as an exemplification of how the survival time is modeled in Survival Analysis. To do that we retrieve the definition of hazard function $\lambda(t)$ from Equation 1.4, and we incorporate the definition of hazard ratio:

$$HR(t) = \frac{\lambda_2(t)}{\lambda_1(t)} \tag{3.1}$$

Which is a way to compare two hazard functions from different individuals of a population. David Cox observed that if the hazard ratio doesn't vary with time $HR(t) = HR$, i.e. the *proportional hazard assumption* holds, then it is possible to estimate the effect parameters without any consideration of the hazard function. From this approach is derived the **Cox model**:

$$\lambda_i(t) = \lambda_0(t) \cdot e^{\beta_1 X_{i1} + \ldots + \beta_p X_{ip}} \tag{3.2}$$

Where $\lambda_0(t)$ is the baseline hazard function, and it can be regarded as the hazard function of an individual whose covariates have all values of zero. The parameters of the Cox model can be interpreted in the following way:

- e^{β_j} represents the hazard ratio for one unit increase in x_j, with all other covariates held constant.
- $\beta_k < 0$ means that if x_j increases the risk (hazard) decreases.
- $\beta_k > 0$ means that if x_j increases the risk (hazard) increases.

However, Cox also noted that the interpretation of the proportional hazards assumption can be quite tricky (Reid, 1994). An alternative to the Proportional Hazard models is the AFT model, which instead assumes that the effect of a covariate is to accelerate or decelerate the life course of a failure by some constant. This appears to be more suitable for mechanical processes.

3.3 Sliding Window

Rather a technique of data preparation, a very common approach when dealing with censored data (and in general with time to an event) is based on establishing a "window" over the series in a way that the temporal axis is disregarded. By resampling the time i.e, establishing periods of duration d, we can reformulate the problem into a binary classification where we ask whether the event will happen or not in the next time window. Which allows for Machine Learning algorithms to be used.

The binary variant of the sliding window approach has been used before in customer churn (XIA and JIN, 2008). And it is useful because of its simplicity; formulations like the customer will stay with us in the next period of duration d are straightforward. In other fields this idea can be adapted into a multiclassification problem or even a regression, yet the inference is still limited to predicting one time window ahead. Forward-looking constructions are especially troublesome.

3.4 Weibull Time To Event

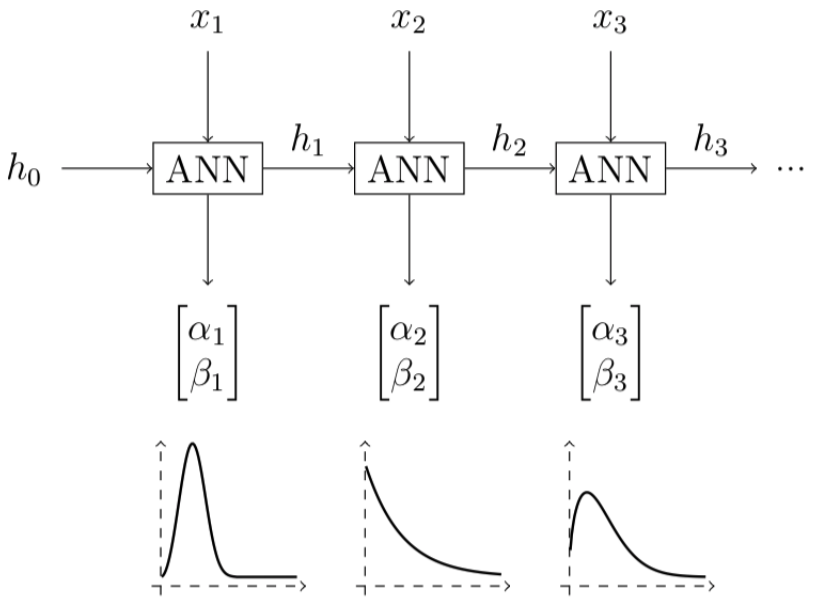


FIGURE 3.2: PDF of uncensored data from Martinsson, 2016

Weibull Time To Event (WTTE-RNN) consists on a framework that uses Recurrent Neural Networks to estimate the two parameters of a Weibull Distribution (chapter 1.2). WTTE-RNN proposes a special objective function (log-likelihood-loss for censored data) that is applicable when we have any or all of the problems of continuous or discrete time, censoring, recurrent events or time series of varying lengths.

The modelling of censored data could be done with a wide variety of distributions: Beta, Gamma, Exponential, Poisson, etc. But Martinsson's work focuses on the Weibull Distribution because it is:

- Empirically feasibe

- Easily discretized
- Unimodal but expressive
- Regularizable
- Numericaly stable

Although the Weibull Distribution has some great properties, the framework can be extended to support other distributions and adapted for multivariate prediction.

3.4.1 Measuring uncertainty

Another interesting property that Martinsson theorizes in his thesis (chapter 2.3), is that given a censored event happening at time t, the mass of the probability density function will be pushed over the left as t comes closer in time. To understand better this idea, let's take a look at the Figure 3.2 and suppose we are trying to predict when there'll be a downtime in the production line. At time n, being $n << t$ a moment of time reasonably far and prior to t, we would expect the pdf of the Weibull Distribution to be rather flat. Meaning the variance is still very high and therefore there is a significant uncertainty about whether the line is going to fail or not. As the failure becomes imminent and therefore n approaches t, the pdf will be pushed over $t - n$, until reaching a narrow peak. Thus it becomes more and more *certain* that the downtime will occur in $t - n$ steps.

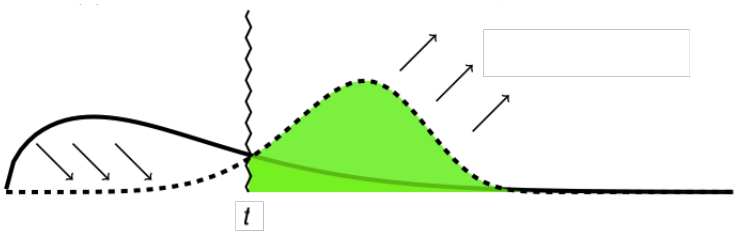


FIGURE 3.3: PDF of right censored data from Martinsson, 2016

In fact, Martinsson also examines the behaviour of the Weibull Distribution with censored data. In the Figure 3.3, the event is right censored, so we know it's going to happen *after* time t. Notice that the measure of uncertainty that we get from an statistic distribution is a very useful property for the Predictive Maintenance use-case. Specially in the context of the pharmaceutical industry, since it is a highly regulated industry. Actions on the line, besides having a cost, have to be justified. Hence it's very important to know when the line is going to fail, and how sure we are about that.

3.4.2 Log-likelihood for censored data

The likelihood $\mathcal{L}(\theta|x)$ is a function of the parameters of a statistical distribution given observed data. The maximum likelihood estimation (MLE) is a method based on the likelihood function to estimate the parameters of a statistical distribution with observed data. So given a statistical model, i.e. a family of distributions $\{f(\cdot;\theta)|\theta \in \Theta\}$, this method selects the parameter values θ that make the data most probable. That's what the RNN will do for us, we just have to figure out the loss function, i.e.

the likelihood.

Finding the maximum of a function often involves taking the derivative of a function and solving for the parameter being maximized. Although the differentiation will be done by the neural network, it is often easier when the function being maximized is the natural logarithm (ln) of the likelihood (log-likelihood). This is because the likelihood function of a collection of statistically independent observations factors into a product of individual likelihood functions. The logarithm of this product is a sum of individual logarithms, and the derivative of a sum of terms is often easier to compute than the derivative of a product.

Since the Time To Failure can only be right censored, we will avoid specifying how the loss would look for left censored data. Assuming that our statistic model $f(\cdot;\theta)$ is the Weibull Distribution and therefore $\theta = (\alpha, \beta)$. Let (t, u) be an observation with u the failure indicator s.t $u = 1$ means that we have an uncensored observation and $u = 0$ a right censored observation, where:

- $f(t)$ is the probability density function
- $F(t)$ is the cumulative density function
- $S(t)$ is the survival function
- $\lambda(t)$ is the hazard function
- $\Lambda(t)$ the cumulative hazard function
- $d(t) = \Lambda(t+1) - \Lambda(t)$

Then we have the following cases:

Continuous case (random variable T)

$$\mathcal{L} = f(t)^u P(T > t)^{1-u} \tag{3.3}$$

$$\mathcal{L} = f(t)^u S(t)^{1-u} \tag{3.4}$$

$$\mathcal{L} = e^{-u\cdot\Lambda(t)} \cdot \lambda(t)^u \cdot e^{-(1-u)\Lambda(t)} \tag{3.5}$$

$$\mathcal{L} = \lambda(t)^u \cdot e^{-\Lambda(t)} \tag{3.6}$$

$$ln(\mathcal{L}) = ln(\lambda(t)^u \cdot e^{-\Lambda(t)}) \tag{3.7}$$

$$ln(\mathcal{L}) = ln(\lambda(t))^u + ln(e)^{-\Lambda(t)} \tag{3.8}$$

$$ln(\mathcal{L}) = u \cdot ln(\lambda(t)) - \Lambda(t) \tag{3.9}$$

The unconstrained optimization problem in the continuous case is then to find w maximizing the log-loss:

$$\underset{w}{maximize} \ln(\mathcal{L}(w, y, u, x)) := \sum_{t=0}^{T} (u_t \cdot [\alpha_t \cdot \ln\left(\frac{y_t}{\beta_t}\right) + \ln(\alpha_t)] - \left(\frac{y_t}{\beta_t}\right)^{\alpha_t}) \tag{3.10}$$

Discrete case (discrete random variable T_d)

$$\mathcal{L}_d = P(T_d = t)^u P(T_d > t)^{1-u} \tag{3.11}$$

$$\mathcal{L}_d = (S(t) - S(t+1))^u \cdot S(t+1)^{1-u} \tag{3.12}$$

$$\mathcal{L}_d = (e^{-\Lambda(t)} - e^{-\Lambda(t+1)})^u \cdot e^{-(1-u)\Lambda(t)} \tag{3.13}$$

$$\mathcal{L}_d = (e^{d(t)} - 1)^u \cdot e^{-\Lambda(t+1)} \tag{3.14}$$

$$ln(\mathcal{L}_d) = ln((e^{d(t)} - 1)^u \cdot e^{-\Lambda(t+1)}) \tag{3.15}$$

$$ln(\mathcal{L}_d) = ln(e^{d(t)} - 1)^u + ln(e)^{-\Lambda(t+1)} \tag{3.16}$$

$$ln(\mathcal{L}_d) = u \cdot ln(e^{d(t)} - 1) - \Lambda(t+1) \tag{3.17}$$

The unconstrained optimization problem in the discrete case is then to find w maximizing the log-loss:

$$\underset{w}{maximize} \ln(\mathcal{L}(w, y, u, x)) := \sum_{t=0}^{T} (u_t \cdot [exp[\left(\frac{y_t + 1}{\beta_t}\right)^{\alpha_t} - \left(\frac{y_t}{\beta_t}\right)^{\alpha_t}] - 1] - \left(\frac{y_t + 1}{\beta_t}\right)^{\alpha_t}) \tag{3.18}$$

Chapter 4

Methodology

4.1 Data set

Initially the intention was to use several real data sets related to Predictive Maintenance where time to failure could be modeled, in order to test the performance of WTTE-RNN (chapter 3.4) in different scenarios. Unfortunately, it's very hard to find open datasets that satisfy these constraints.

At the end, I decided to focus on the Turbofan Engine Degradation Simulation Data Set (Saxena and Goebel, 2008). Although this is a **simulated** dataset, it comes from one of the simulators at NASA Ames, CA. In particular, this simulator is called C-MAPSS, which stands for Commercial Modular Aero-Propulsion System Simulation, and it is a tool for the simulation of realistic large commercial turbofan engine data.

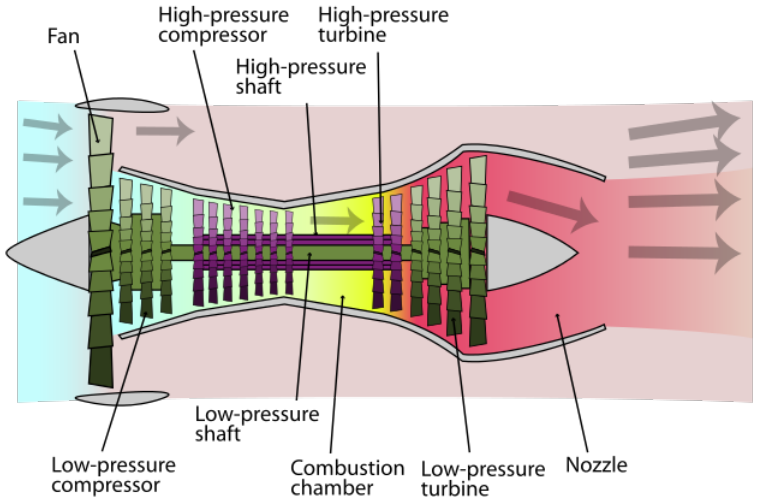


FIGURE 4.1: Turbofan operation diagram from Aainsqatsi, 2008

There is a paper about the generation of the data set (Saxena et al., 2008), basically it consists of multiple multivariate time series. Each time series is from a different engine i.e., the data can be considered to be from a fleet of engines of the same type. Each engine starts with different degrees of initial wear and manufacturing variation which is unknown to the user. This wear and variation is considered normal, i.e., it is not considered a fault condition. There are three operational settings that have a substantial effect on engine performance. These settings are also included in the data. The data is contaminated with sensor noise.

The data is provided as a zip-compressed text file with 26 columns of numbers, separated by spaces. Each row is a snapshot of data taken during a single operational cycle, each column is a different variable. The columns correspond to:

- Unit number
- Time (cycles)
- Operational setting 1-3
- Sensor measurement 1-21

cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
173	0.316092	0.666667	0.0	0.0	0.530120	0.428385	0.726199	0.0	...	0.302772	0.514706	0.111983	0.708734	0.0	0.583333	0.0	0.0	0.348837	0.393123
174	0.494253	0.333333	0.0	0.0	0.430723	0.627207	0.680790	0.0	...	0.326226	0.485294	0.099236	0.472105	0.0	0.416667	0.0	0.0	0.333333	0.195388
175	0.505747	0.666667	0.0	0.0	0.722892	0.703074	0.679777	0.0	...	0.360341	0.500000	0.133244	0.631397	0.0	0.500000	0.0	0.0	0.341085	0.363297
176	0.385057	0.250000	0.0	0.0	0.496988	0.462830	0.683322	0.0	...	0.236674	0.470588	0.123129	0.617160	0.0	0.500000	0.0	0.0	0.449612	0.490058
177	0.281609	0.250000	0.0	0.0	0.777108	0.675387	0.704591	0.0	...	0.330490	0.514706	0.104087	0.588688	0.0	0.666667	0.0	0.0	0.379845	0.377106
178	0.494253	0.666667	0.0	0.0	0.653614	0.747547	0.715733	0.0	...	0.321962	0.470588	0.085767	0.585225	0.0	0.583333	0.0	0.0	0.395349	0.390500
179	0.632184	0.250000	0.0	0.0	0.496988	0.469152	0.796759	0.0	...	0.360341	0.441176	0.087212	0.653328	0.0	0.583333	0.0	0.0	0.488372	0.406794
180	0.362069	0.500000	0.0	0.0	0.713855	0.628515	0.588960	0.0	...	0.245203	0.588235	0.116782	0.572528	0.0	0.583333	0.0	0.0	0.558140	0.155758
181	0.465517	0.916667	0.0	0.0	0.671687	0.559625	0.648042	0.0	...	0.371002	0.514706	0.076633	0.585987	0.0	0.750000	0.0	0.0	0.155039	0.374206
182	0.442529	0.750000	0.0	0.0	0.903614	0.676695	0.748987	0.0	...	0.377399	0.544118	0.101816	0.734513	0.0	0.750000	0.0	0.0	0.224806	0.392847
183	0.505747	0.583333	0.0	0.0	0.611446	0.570961	0.620864	0.0	...	0.289979	0.514706	0.092166	0.689496	0.0	0.666667	0.0	0.0	0.325581	0.325877
184	0.522989	0.750000	0.0	0.0	0.861446	0.749945	0.848582	0.0	...	0.230277	0.588235	0.080865	0.724894	0.0	0.583333	0.0	0.0	0.224806	0.166252
185	0.545977	0.583333	0.0	0.0	0.780120	0.706780	0.711512	0.0	...	0.240938	0.573529	0.070957	0.667564	0.0	0.583333	0.0	0.0	0.286822	0.242751
186	0.655172	0.250000	0.0	0.0	0.692771	0.525834	0.743585	0.0	...	0.296375	0.544118	0.121323	0.761062	0.0	0.750000	0.0	0.0	0.255814	0.223971
187	0.229885	0.500000	0.0	0.0	0.635542	0.459124	0.759959	0.0	...	0.179104	0.588235	0.081175	0.757599	0.0	0.666667	0.0	0.0	0.217054	0.259597
188	0.114943	0.750000	0.0	0.0	0.765060	0.683235	0.684166	0.0	...	0.234542	0.514706	0.091599	0.753367	0.0	0.666667	0.0	0.0	0.286822	0.089202
189	0.465517	0.666667	0.0	0.0	0.894578	0.547853	0.772451	0.0	...	0.189765	0.661765	0.090670	0.744132	0.0	0.583333	0.0	0.0	0.263566	0.301712
190	0.344828	0.583333	0.0	0.0	0.731928	0.614345	0.737677	0.0	...	0.287846	0.691176	0.065229	0.759523	0.0	0.833333	0.0	0.0	0.271318	0.239299
191	0.500000	0.166667	0.0	0.0	0.641566	0.682799	0.734639	0.0	...	0.187633	0.617647	0.075704	0.740669	0.0	0.500000	0.0	0.0	0.240310	0.324910
192	0.551724	0.500000	0.0	0.0	0.701807	0.662089	0.758778	0.0	...	0.296375	0.647059	0.056714	0.717199	0.0	0.666667	0.0	0.0	0.263566	0.097625

FIGURE 4.2: Example of engine 1 data for the last 20 cycles

The data is partitioned in four subsets: FD001 with 1 operating conditions and 1 failure mode, FD002 with 6 operating conditions and 1 failure mode, FD003 with 1 operating condition and 2 failure modes, and FD004 with 6 operating conditions and 2 failure modes. The 2 failure modes correspond to the fan degradation and the high-pressure compressor degradation (check Figure 4.1), whereas the 6 operating conditions are a combination of altitude, flight speed and TRA.

The engine is operating normally at the start of each time series, and develops a fault at some point during the series. If we observe Figure 4.2, we can see that the engine 1 was monitored until the cycle 192 when the failure occurred. The objective is to predict the number of remaining operational cycles before the failure in each cycle e.g., we would like to know at cycle 172 that there are 20 cycles left to the failure.

4.2 Validation Process

In the original training set of the Turbofan Engine Degradation Simulation Data Set, the fault grows in magnitude until system failure. Whereas in the test set the time series ends some time prior to system failure. Since the time series are "interrupted"

and are not observed until the end, we will additionally separate 20% of the original train data set as a validation data set . This corresponds to 20 batches of approximately 200 cycles each , since there are 100 engines in the training set and 100 more in the test set. The split will be done with a random seed (42) for reproducibility purposes. We call *batch* the sequence of observed cycles per engine.

Besides the loss function employed at each experiment, there will be a set of metrics common in each trial. Notice that the models based on the WTTE-RNN architecture (chapter 3.4) predict the 2 parameters of a Weibull Distribution (chapter 1.2). With this statistic model the expected value of the number of remaining cycles to the failure, formally called Remaining Useful Life (RUL), can be estimated in different ways (next chapter). Therefore there will be some models evaluated several times, this will be useful to additionally compare these methods to compute the RUL.

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(y_i - \tilde{y}_i)^2} \tag{4.1}$$

$$MAE = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - \tilde{y}_i) \tag{4.2}$$

The Root Mean Squared Error ($RMSE$) and the Mean Absolute Error (MAE) are standard metrics for regressions. We refer as error to the difference between the observed values y and the predicted values $\tilde{y}$.

$$R^2 = 1 - \frac{\Sigma_{i=1}^{n}(y_i - \mu)^2}{\Sigma_{i=1}^{n}(y_i - \tilde{y}_i)^2} \tag{4.3}$$

Another common metric is the Coefficient of Determination, usually called R squared (R^2). It computes the mean of the observed values μ and it measures "how well" the regression predictions approximate the real data points. It usually ranges from 0 to 1 but values outside that range can occur if the model fits worse than a horizontal hyperplane.

$$s = \begin{cases} \Sigma_{i=1}^{n} e^{-\left(\frac{\tilde{y}_i - y_i}{10}\right)} & (\tilde{y}_i - y_i) < 0 \\ \Sigma_{i=1}^{n} e^{\left(\frac{\tilde{y}_i - y_i}{13}\right)} & (\tilde{y}_i - y_i) \geq 0 \end{cases} \tag{4.4}$$

In the paper (Saxena et al., 2008) the authors introduce an score s where the penalty grows exponentially with increasing error. Therefore in this scoring technique late predictions are more heavily penalized than early predictions, which is an interesting feature for Predictive Maintenance. Even so, it's difficult to interpret since it's likely to produce very high values for relatively small errors. Thus it will not be included in the evaluation, instead our aim is to include $RMSE$, MAE and R^2 as metrics for the models.

4.3 Software

Deep Learning is receiving a lot of attention these days and consequently there are many tools in continuous development. It's hard to keep up with the good work that is being done, in order to explore all the frameworks that are popular among the developers. Instead, we will rely on Google's project TensorFlow™and it's API Keras.

Pos	Name	Description	Language	Stars
1	tensorflow	Computation using data flow graphs for scalable machine learning	C++	105193
2	keras	Deep Learning for humans	Python	31664
3	opencv	Open Source Computer Vision Library	C++	26216
4	caffe	Caffe: a fast open framework for deep learning.	C++	24886
5	TensorFlow-Examples	TensorFlow Tutorial and Examples for Beginners with Latest APIs	Jupyter Notebook	23594
6	machine-learning-for-software-engineers	A complete daily plan for studying to become a machine learning engineer.	None	19241
7	deeplearningbook-chinese	Deep Learning Book Chinese Translation	TeX	18754
8	Deep-Learning-Papers-Reading-Roadmap	Deep Learning papers reading roadmap for anyone who are eager to learn this amazing tech!	Python	18143
9	pytorch	Tensors and Dynamic neural networks in Python with strong GPU acceleration	C++	17157
10	awesome-deep-learning-papers	The most cited deep learning papers	TeX	15409
11	Detectron	FAIR's research platform for object detection research, implementing popular algorithms like Mask R-CNN and RetinaNet.	Python	15242
12	CNTK	Microsoft Cognitive Toolkit (CNTK), an open source deep-learning toolkit	C++	14815

FIGURE 4.3: Top Deep Learning projects from Badry, 2018

In the Figure 4.3 we have list of Deep Learning projects from Github updated in July 2018. As we can see TensorFlow™is the most popular, since besides an open source software library it's conceived for high performance numerical computation. Additionally its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs). On the other hand, Keras is a high-level neural networks API written in Python capable of running on top of TensorFlow, CNTK, or Theano. More importantly, it is focused on enabling **fast experimentation**.

To achieve it we will complement these frameworks with a usable and interactive working environment called Jupyter Notebook, which is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. An extension of this idea is Colaboratory, another Google project that provides a free running environment with GPU support.

Chapter 5

Experimentation

Finally we present all the experimentation that has been done in this thesis. Even though the last sections were quite theoretic this chapter will be presented from a practical perspective, not only showing the results obtained but also focusing in the relevant parts of the code implementation. That's why all the code will be posted in my personal Github Account complemented with some explanations and visualizations thanks to Jupyter Notebook. Thus, I strongly recommend to visit that resource if one wants a hands-on approach, the easiest way to start playing with the notebook is by opening it with Collaboratory.

Just to recapitulate, the scope of this thesis is to use Recurrent Neural Networks (1.3) in order to predict the time to an event. Which framed in the context of my recent professional experience translates into predicting the time to **failure**. In particular we want to target the WTTE-RNN (3.4) architecture proposed by Martinsson from a realistic use-case scenario, validating the theoretic work of his thesis and exploiting the benefits of the Weibull Distribution (1.2) for Predictive Maintenance (2.2).

5.1 Data preparation

Maybe one of the more important decisions to take when creating a model is how to represent the given information in the best way possible. To do it it's necessary to identify with detail which is the exact question that the model is supposed to answer. In our case we narrowed down the possibilities by specifying that we aim to predict the Remaining Useful Life (RUL) **in each cycle of an engine**. So if we were to start running one of the engines, we would like to know from the first instance when it is expected to fail. Notice that with this data set we could turn things around by changing the question that we want to ask to the model. For instance we could train a binary classifier to know whether the engine is going to fail in the following n cycles. More interesting maybe it is to know the probability that the failure is going to occur. Or perhaps we want to find out if the engine is going to survive after a given time t or not. Further on we will see that by predicting the parameters of a Weibull Distribution we are able to answer many questions.

Even with these constraints about the purpose of the model, there are a few ways to prepare the data and many alternatives when implementing the Recurrent Neural Network. If we imagine the data set as a 3D tensor, we distinguish two main approaches when preparing the data for this problem:

1. **Rolling Window**: It is based on setting a look-back period i.e., a fixed number of previous cycles to look at. Therefore choosing the right look-back period becomes another important decision for the model. For the first cycles that do not

have enough previous information we can apply left padding[1] to incorporate them to the training. It's important to say that RNN layers have internal states about how a sequence is evolving as it steps forward. Windows eliminate the possibility of learning long sequences, limiting all sequences to the window size. The shape of the data is:
(total number of cycles, look-back period, number of features)

2. **Batch Mode**: In this case we don't restrict to a fixed number of past "time steps", but instead we use all the previous information of the batch. The advantage here is that the state of the RNN layer is preserved for each batch, and it is also easy to shuffle the batches in each epoch of the training. To implement this with Keras we have to apply right padding to each sequence so they all have the same length, therefore the resulting shape is:
(total number of batches, batch of maximum length, number of features)

Among the possible ways to implement an RNN that we observe in Figure 5.1 we highlight *many-to-one* and *many-to-many*. The main difference is that in many-to-one we exclusively return the output state of the last unit h_t, whereas in many-to-many we return the states of all the units $h_t, h_{t-1}, h_{t-2}, ..., h_{t-n}$.

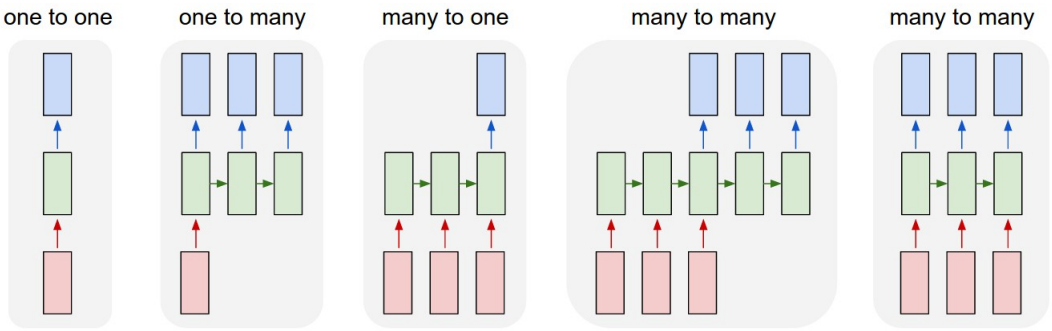


FIGURE 5.1: Setups for RNNs from Karpathy, 2015

Because in the Rolling Window case we treat each sequence of look-back period length independently, many-to-one is the inherent implementation. In Keras there is a workaround to try to preserve the state of the RNN unit between sequences that belong to the same batch. It is an infamous API called *stateful* that it's not trivial to use at all. We will not present the work done with this type of data reshaping, there are others (@daynebatten, @gm-spacagna) that have already done the effort.

There are ways to implement many-to-one for the Batch Mode but these are less intuitive and more prone for errors. Instead **we will focus on the many-to-many implementation for the Batch Mode**, which feels like the more natural option for the CMAPSS data set. The Keras implementation of this mode involves masks, sample weights and other techniques that will be explained next.

[1] In Machine Learning *padding* is a technique to extend a sequence to a new longer desired length by using dummy values.

5.2 Baseline

We establish a baseline model that uses Recurrent Neural Networks to predict the Remaining Useful Life. The objective is to select a model that obtains reasonable results with the validation process that has been established. Then we will adapt the model to the WTTE-RNN architecture to compare the results.

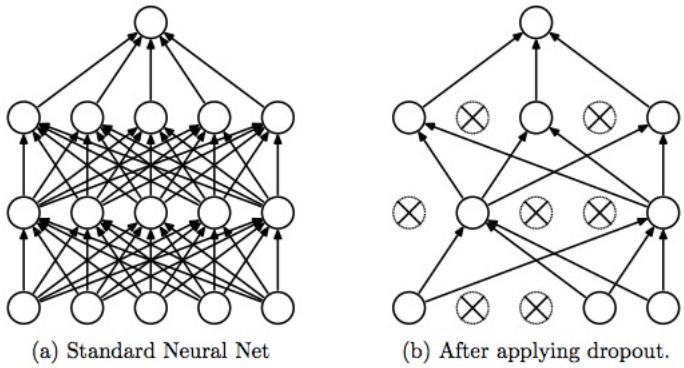
Layer (type)	Output Shape	Param #
masking_1 (Masking)	(None, 362, 25)	0
lstm_1 (LSTM)	(None, 362, 100)	50400
lstm_2 (LSTM)	(None, 362, 50)	30200
time_distributed_1 (TimeDist	(None, 362, 1)	51

FIGURE 5.2: Baseline network topology

The model is based on a popular Github Repository (Griffo, 2018) and it basically consists of two stacked LSTM layers with 100 and 50 units respectively. On top there is a masking layer that will mask the padding of each engine, this will be propagated layer-by-layer and eventually applied to the loss. Therefore the padding placeholders will be skipped in the LSTM and ignored in the loss. Finally there is a time-distributed layer, which is a keras object that applies the same dense layer to every time step of the LSTM unit. An important detail with the many-to-many approach is that it was necessary to use an **exponential activation function** e^x .

5.2.1 Regularization

As it can be seen in the Figure 5.2 there is a large number of parameters (80k) in comparison with the number of samples (13k coming from 100 engines) so it's easy to "learn by hard" the problem. To avoid overfitting[2] we include two main regularization techniques *Dropout* and *Early Stopping*.



(a) Standard Neural Net (b) After applying dropout.

FIGURE 5.3: Dropout representation from Srivastava et al., 2014

[2] *Overfitting* happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data.

Dropout (Srivastava et al., 2014) is a simple but very effective idea based on randomly ignoring neurons with a certain probability. It turns out that Dropout can be interpreted as learning many different neural networks, and it's well known that better results can generally be obtained when using Ensemble Learning (i.e., multiple independent models). Since each of these models has a different overfitting, it is prevented by taking an average. The problem is that RNN have connections inside the layer, and if we cancel RNN units without paying attention to the connections between these units the noise will be amplified for long sequences drowning the signal (Zaremba, Sutskever, and Vinyals, 2014).

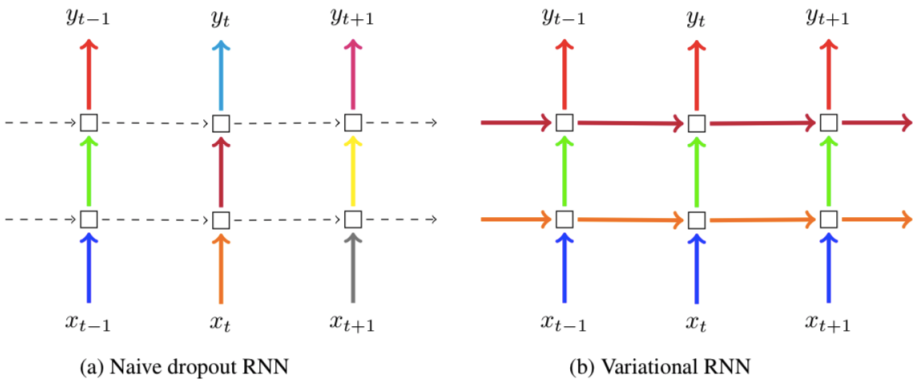


FIGURE 5.4: Recurrent dropout from Gal and Ghahramani, 2016

Therefore we incorporate *Recurrent Dropout* (Gal and Ghahramani, 2016) to the baseline, the theory is quite complex and it is based on interpreting dropout as a variational approximation to the posterior of a Bayesian neural network. In practice it means masking as well the connections between RNN units in a particular way. One can get the intuition by checking Figure 5.4, coloured connections represent dropped-out inputs, with different colours corresponding to different dropout masks.

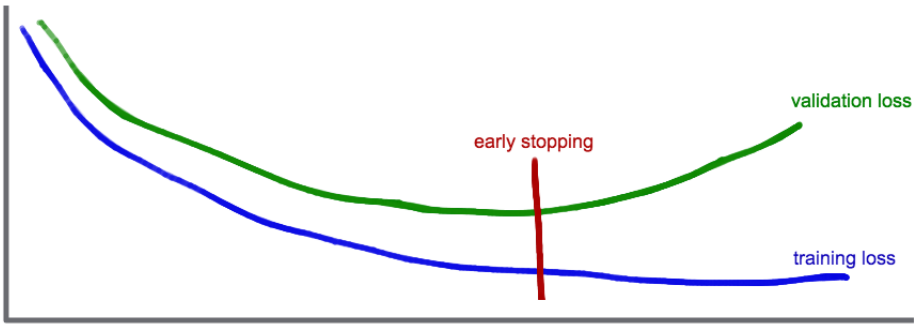


FIGURE 5.5: Training loss and validation loss when overfitting

As for Early Stopping we simply monitor the validation loss and when we detect that it's increasing we stop the training. This is typically done by setting a patience period p, if the validation loss doesn't overcome the last best score in p epochs the execution is automatically terminated.

5.2.2 Results

We trained the model with Collaboratory (GPU) for 318 epochs (14 secs/epoch) with an Early Stopping patience of 30 epochs over the validation loss. We use the RMSProp optimizer with learning rate set to 0.001 since it's suggested for Recurrent Neural Networks in the Keras documentation. The data is scaled (min-max) and it's organized in batches (with batch size = 16) so the state of the RNN units is preserved, the engines are shuffled in each epoch.

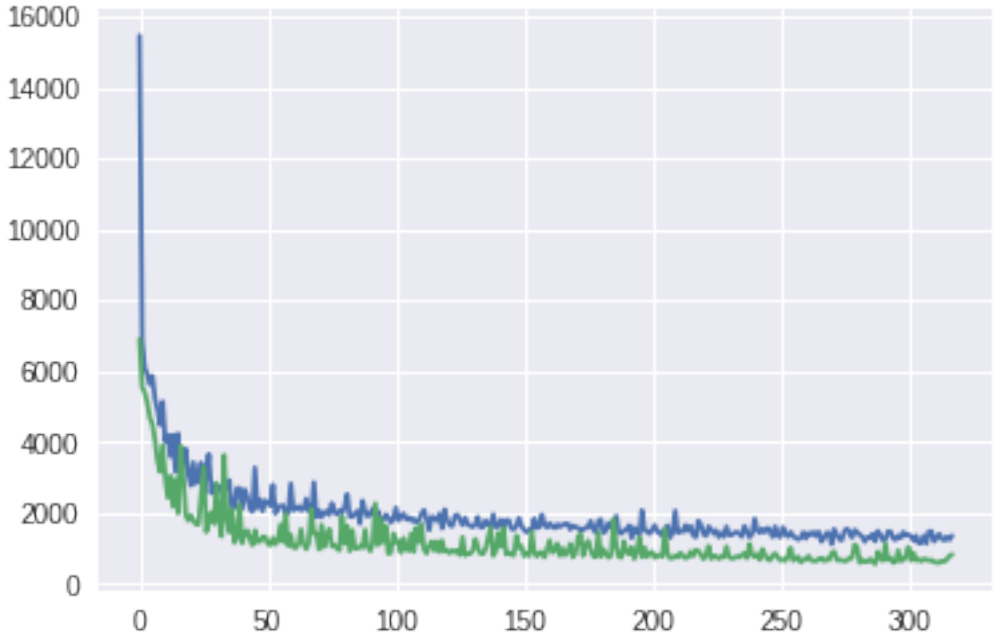


FIGURE 5.6: Training loss (blue) and validation loss (green)

In the tests done with the Rolling Window approach we obtained successful results using a linear function as an activation function for the last dense layer of the model. However, this produces disastrous results for the Batch Mode with the many-to-many implementation, where the model fits a horizontal line across all the engines. Just out of curiosity we implemented the many-to-one for the batch mode, which implies removing the time-distributed dense layer with 1 neuron, to substitute it with a dense layer with a number of neurons equal to the batch of maximum length.

TABLE 5.1: Evaluation of the baseline

Set	MAE	$RMSE$	R^2
Train	21.19	33.57	0.766
Val.	17.36	23.98	0.866
Test	27.03	37.41	0.598

In this case the model learned the RUL slope of the average engine and was producing the same output for all the engines, what seems like a case of overfitting despite

of the regularization techniques being used. Finally it was solved with an exponential activation function. Probably there are other alternatives that would also work, we will leave that as future work (chapter 6.1).

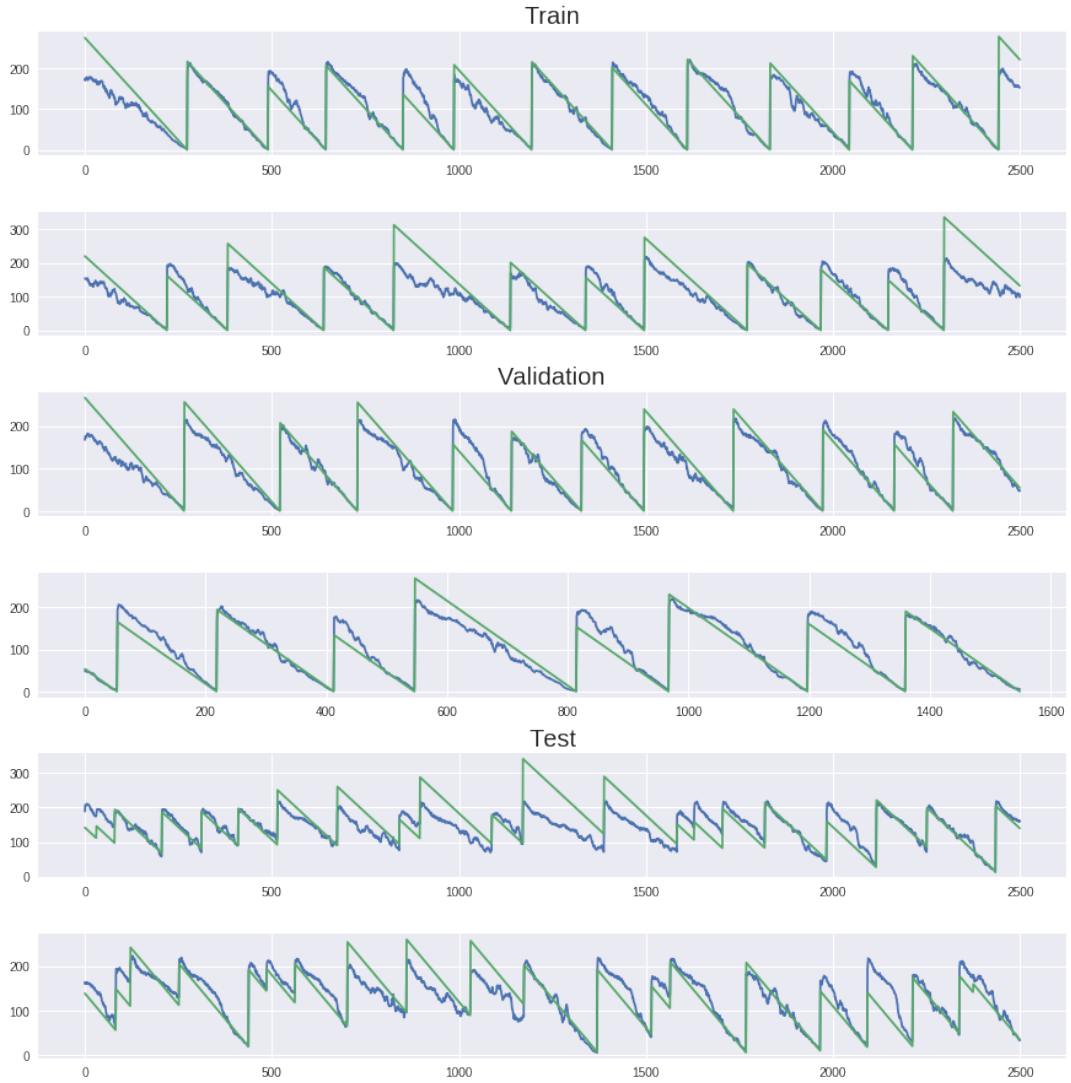


FIGURE 5.7: Predicted RUL blue vs real RUL green

In short, we can observe the results of the evaluation (without the padding) in the table 5.1. Also we can take a loot at Figure 5.7, which plots the predicted RUL of a selection of engines from the sets of train, validation and test. It seems that the baseline model it's learning with significant detail most of the sequences, with a few exceptions on the test set.

One can also intuit that the model is usually more precise in the 50-ish last cycles of the sequence. In the next chapter the baseline architecture will be adapted to the WTTE-RNN model, and we will try to understand what is the behaviour of the Weibull Distribution across the cycles of an engine.

5.3 Adapting to WTTE-RNN

The content of the CMAPSS data set it's *discrete*, as it was explained in the previous chapter (4.1) the degradation of turbofan engines is measured per cycle. Hence we just need to refer to the Equation 3.18 to implement the log-likelihood loss, in Keras it would look like this:

```
def weibull_loglik_discrete(alpha, beta, y, u, epsilon=K.epsilon()):
    hazard0 = K.pow(K.div(y + epsilon, beta), alpha)
    hazard1 = K.pow(K.div(y + 1., beta), alpha)
    return K.mul(u, K.log(K.exp(hazard1 - hazard0) - 1.0)) - hazard1
```

Additionally we include an epsilon to avoid numerical instability, notice also that $u_t = 1 \ \forall t$ since we know the exact time of the failure for all the engines of the CMAPSS data set. Formally we say that all the samples are uncensored (chapter 1.1). Still this is a very powerful attribute from the WTTE-RNN architecture. Besides the loss function we need to consider the activation functions for the α and β neurons, which should be in the last layer of the network. Initially Martinsson suggested in his thesis (chapter 4.2.1) a softplus activation $ln(1 + e^x)$for the shape parameter α and the scale parameter β, but later rectified into a sigmoid $\frac{1}{1+e^{-x}}$ for the shape parameter α and a exponential e^x for the scale parameter β.

```
def weibull_activation(alpha, beta):
    alpha = K.sigmoid(alpha)
    beta = K.exp(beta)
    return alpha, beta
```

Martinsson has encountered exploding gradients in some cases mainly involving censored data, the sigmoid function has nice implicit regularization features such as the maximum value for α. Regarding the exponential function this seems to converge faster in real life, which makes sense due to the logarithmic effect of the softplus activation, still Martinsson recalls the importance of initializing β around it's scale. That's why he has pursued in the creation of a python package named wtte that it's installable through pip. This tool includes updated versions of the loss functions for discrete and continuous data, plus additional regularization techniques such as initialization, maximum values and gradient clipping.

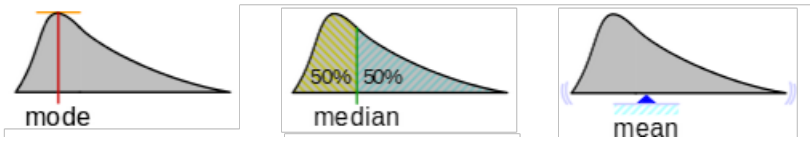


FIGURE 5.8: Mean, median and mode from Wikipedia

After successfully adapting the baseline to the WTTE-RNN architecture, the next question is how to compute the expected value of the RUL given the parameters α and β. The mode (Equation 1.11) seems to be the best choice, specially taking into account that the Weibull Distribution can be skewed[3]. Anyways we additionally compare the prediction of the RUL with the mean (Equation 1.9) and the median (Equation 1.12).

[3]In probability theory and statistics, *skewness* is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean.

5.3.1 Results

The model was trained under the same conditions than the baseline (chapter 5.2.2). Although the Early Stopping patience was also set to 30 epochs, the training was running for 357 epochs at 12 secs/epoch. The times might be a little different because Collaboratory assigns resources depending on the demand i.e., the computing power is shared among users.

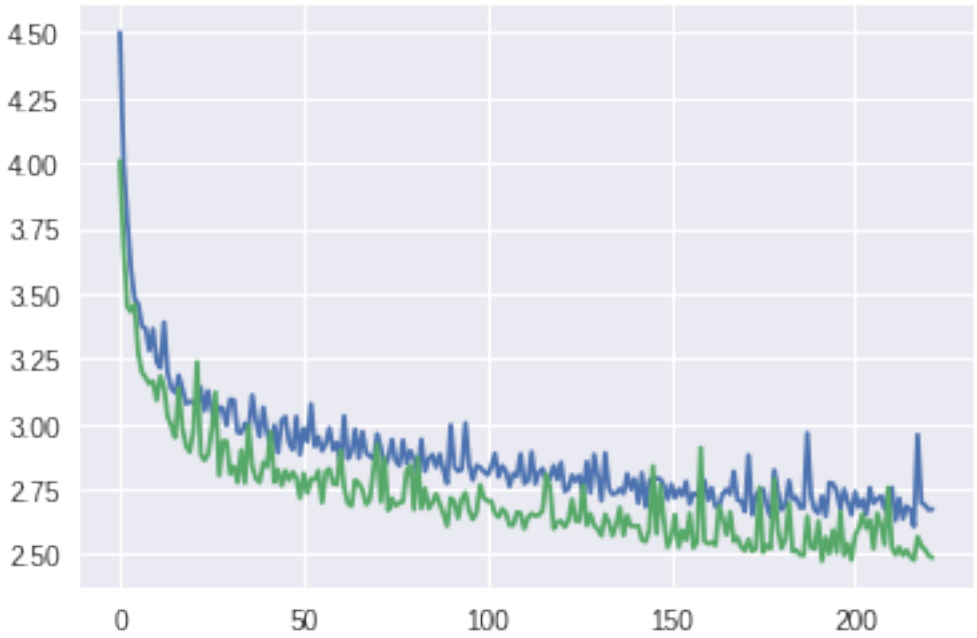


FIGURE 5.9: Training loss (blue) and validation loss (green)

With the wtte package we set a maximum of 10 to the shape parameter α and initialize the scale parameter β around the mean of the RUL, under Martinsson suggestion. This seems to make a significant difference to obtain competitive results, otherwise the model converges slower.

TABLE 5.2: Evaluation of the WTTE-RNN (mode, median and mean)

	Mode			Median			Mean		
Set	*MAE*	*RMSE*	R^2	*MAE*	*RMSE*	R^2	*MAE*	*RMSE*	R^2
Train	21.53	34.69	0.750	21.05	33.51	0.767	20.94	33.14	0.772
Val.	17.94	26.48	0.836	17.79	25.48	0.848	17.79	25.26	0.851
Test	27.46	38.59	0.572	26.72	37.49	0.596	26.51	37.22	0.602

We remove the left padding to evaluate the sequences. Surprisingly the mode performs worst than the other methods, being the mean the best expected value for the RUL in all the sets. The differences are relatively small which suggests that the Weibull Distribution stays symmetric during the engine life-cycle. Still the Mode is the value most likely to be sampled from the distribution, perhaps the reason is that the mode can produce NaN values when it approaches an infinite spike, and it's set to zero when $\alpha \leq 1$. At the end **we will use the mean as the expected value of the RUL** for visualizations and comparatives.

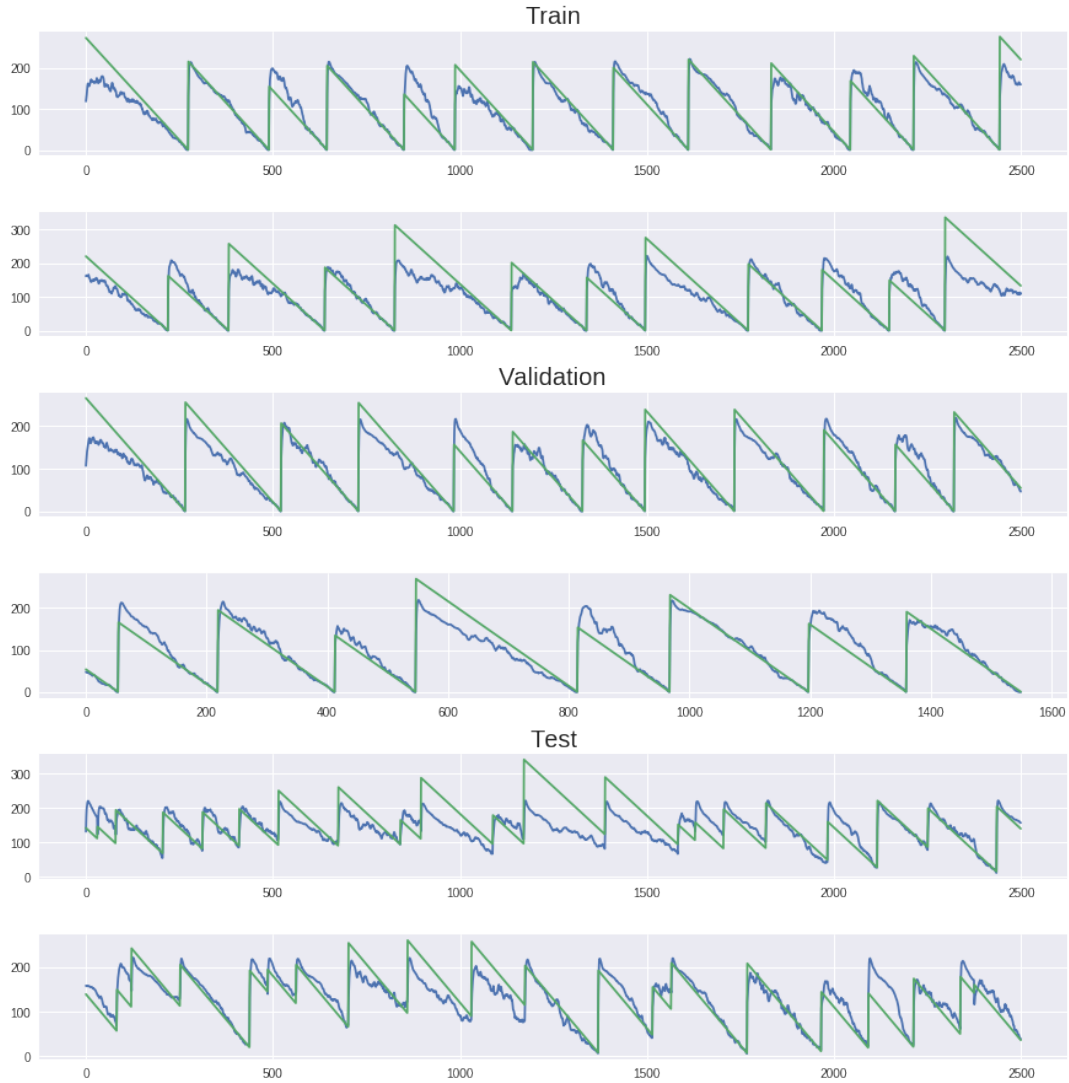


FIGURE 5.10: Predicted RUL blue vs real RUL green

In comparison with the baseline (table 5.1), the WTTE version performs slightly better in the test set (table 5.2). Nevertheless the difference is minimum and we can't say the model is outstandingly better by means of the evaluation metrics (chapter 4.2). Instead we have to center the attention in the attributes of the predicted Weibull Distribution. To begin with, we can examine the shape of the pdf of the Weibull Distribution as the failure becomes imminent. If we remember the theory from Martinsson thesis (chapter 3.4.1), we expect the mass of the pdf to be pushed over the RUL as the event comes closer in time. In order to observe this phenomenon, we plot the pdf across time for a selection of engines from train, validation and test set.

In Figure 5.11 there are 3 randomly selected engines from each of the sets. The pdfs of the whole sequence are overlapped, ranging from blue to red through the cycles. As we explained in the previous chapter 4.2, the test set is right censored which makes the plots more exemplifying. If we observe these from right to left, we can get a "step by step" intuition of what's happening with the pdf.

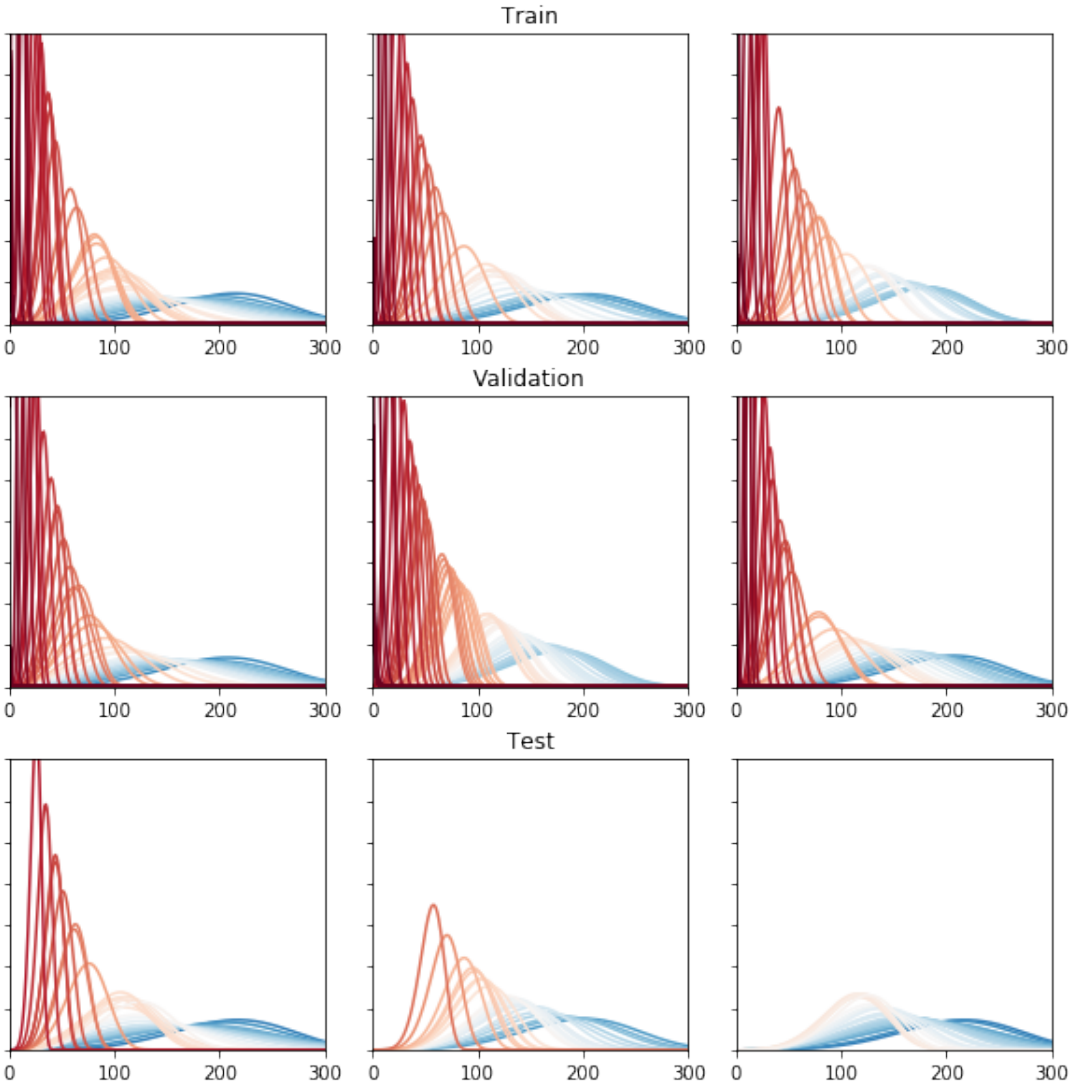


FIGURE 5.11: PDF of the Weibull Distribution during the lifetime of an engine

At the end, it can be seen that the Weibull Distribution behaves as expected. This opens up to many possibilities to complement the expected value of the RUL. For instance, the variance (Equation 1.10) or the standard deviation are straightforward measures that we can get from the distribution. The Survival function is another interesting measure since it computes the probability that the engine survives the predicted RUL. We also have the possibility to compute the Confidence Interval of the Weibull Distribution, which loosely speaking[4] quantifies the level of confidence that the expected value of the RUL lies in the interval. We generated some illustrative gifs incorporating these ideas.

[4]More strictly speaking, the confidence level represents the frequency the proportion of possible confidence intervals that contain the true value of the unknown population parameter (i,e. the expected value of the RUL).

5.3.2 GRU variant

Gated Recurrent Units (chapter 1.3.2) are a very popular alternative to LSTMs (chapter 1.3.1) because they have less parameters and therefore are faster in practice. We did the exercise to replace the LSTM layers by GRU layers with the same number of units, 100 for the first layer and 50 for the second. In this case the model has 60k parameters instead of 80k.

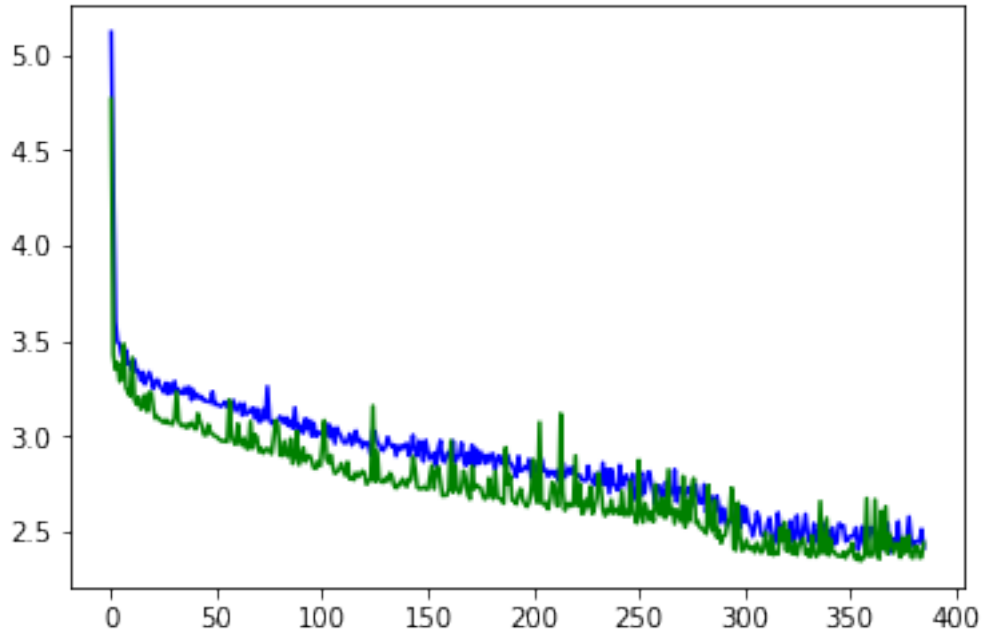


FIGURE 5.12: Training loss (blue) and validation loss (green)

In this case the model was trained locally with a CPU of 1,8 GHz Intel Core i5 and an 8 GB 1600 MHz DDR3 ram memory. Each epoch took around 4 seconds, which is approximately 3 times faster than the LSTM baseline. Having said that, gradients were exploding within the first 50 epochs. It was necessary to incorporate another tool from the wtte package, which is a scale factor for both the shape and scale parameters of the predicted Weibull Distribution. It was set to 0.25 in this experiment and it meant a significant difference since the model was able to run for 386 epochs without being terminated for exploding gradients.

TABLE 5.3: Evaluation of the GRU variant

Set	MAE	$RMSE$	R^2
Train	15.94	25.90	0.861
Val.	18.30	27.46	0.824
Test	25.82	36.70	0.613

An Early Stopping patience of 30 epochs was also used in this case. If we take a look at the table 5.3 we can see that the model fits better the train set than the validation set, and this turns out to improve the scores in the test set. In conclusion, the GRU variant is much faster and obtains good results but it's less stable than the LSTM version and it needs more attention.

Chapter 6

Discussion

6.1 Future Work

Feeling more comfortable with the theory behind the WTTE-RNN framework, and after successfully implementing a baseline model, there is a tremendous amount of possible ideas that are worth a try. From tweaking every component of the proposal to designing complex experiments, these are some of the ideas we came up with:

- **Train on test data**: Since the log-likelihood loss supports censored data, it would be interesting to train a model with the test data from the CMAPSS data set (chapter 4.1). All of the sequences are "interrupted" in the test set, which makes it harder for the model. We did a couple of trials but we didn't manage to obtain successful results.
- **Missing data**: One of the advantages of predicting a statistic distribution is the measure of uncertainty. Other than a mask, which is simply ignored by every layer of the model, we could try to put "holes" in the data to observe how does the pdf of the Weibull Distribution behave. We would expect a variance increase on those holes.
- **Measurement Noise**: Sensors degrade with the time, up to the point that they can produce aberrant or missing values. But during that process usually two things change: the amplitude of the signal gets wider, and the slope decreases. It would be interesting to experiment with new and old sensors, to observe the effect in the variance of the Weibull Distribution.
- **Weak learners**: A little bit out of the scope of the thesis but an interesting idea that Martinsson shared with me, consists in training many weak RNNs instead of a single stacked RNN. As a sort of boosting technique.
- **Different distributions**: Why the Weibull? Perhaps there are other alternatives that work better for particular cases. Maybe the beta distribution would be interesting to try.
- **Regularization**: Although Martinsson already has done great work on this line, I'm sure there are still things to improve. Specially taking into consideration that exploding gradients are guaranteed above a moderate scale ($\simeq 200$).
- **Multivariate support**: In some cases we are interested in compound events i.e., events defined by the realization of a set of events. It would be interesting to extend the model for a multivariate target.

6.2 Conclusions

We have focused on the many-to-many implementation of the batch mode (chapter 5.1), and we have successfully implemented a baseline model to predict the Remaining Useful Life (RUL) using LSTM Networks (chapter 1.3.1). We have managed to adapt this baseline to the WTTE-RNN framework (chapter 3.4) proposed by Egil Martinsson, and we have experimented with the properties of the Weibull Distribution (chapter 1.2). Additionally, we have implemented a variant of the WTTE model using a GRU network (chapter 1.3.2). As a result, we have evaluated (chapter 4.2) three different models:

- Baseline (chapter 5.2.2)
- Baseline WTTE (chapter 5.3.1)
- Baseline WTTE GRU (chapter 5.3.2)

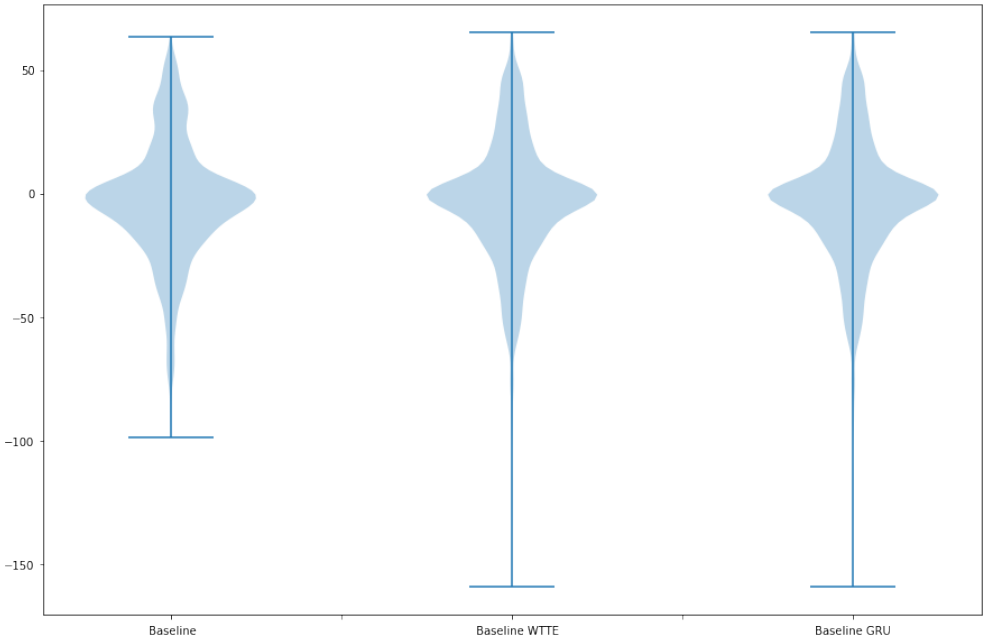


FIGURE 6.1: Violin plot of the error $(\tilde{y} - y)$ for the three models

In terms of the validation loss, the baseline model is obtaining better results. On the other side,v is the worst on the test set. Since the validation set was generated from the original training set, and it is used as stopping criteria for the training, it looks like overfitting the training set benefits on the validation score but penalizes on the test evaluation. In any case, **we have seen that the WTTE-RNN model is just as good as a regular regressor, but has many interesting attributes that are relevant for the Predictive Maintenance context**.

Even so, if we were able to develop a model in a pharmaceutical company capable of making reliable predictions about the manufacturing line, the plant should justify any decision made in front of certain organizations that are not familiar with Deep Learning. In conclusion, we believe that the model is very interesting for many industries, but in the case of the pharma industry there is still a long way to go.

Bibliography

Aainsqatsi, K. (2008). *Turbofan operation*. URL: https://commons.wikimedia.org/wiki/File:Turbofan_operation.svg#/media/File:Turbofan_operation.svg.

Amruthnath, Nagdev and Tarun Gupta (2018). "Fault Class Prediction in Unsupervised Learning using Model-Based Clustering Approach". In:

Badry, Mahmoud (2018). *Top Deep Learning*. URL: https://github.com/mbadry1/Top-Deep-Learning.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. URL: http://www.aclweb.org/anthology/D14-1179.

Gal, Yarin and Zoubin Ghahramani (2016). "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., pp. 1027–1035. ISBN: 978-1-5108-3881-9. URL: http://dl.acm.org/citation.cfm?id=3157096.3157211.

Gorski, A. C. (1968). "Beware of the Weibull euphoria". In: *IEEE Transactions on Reliability* R-17.4, pp. 202–203. ISSN: 0018-9529. DOI: 10.1109/TR.1968.5216949.

Griffo, Umberto (2018). "Predictive Maintenance using LSTM". In: *GitHub repository*. URL: https://github.com/umbertogriffo/Predictive-Maintenance-using-LSTM.

Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever (2015). "An Empirical Exploration of Recurrent Network Architectures". In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, pp. 2342–2350. URL: http://dl.acm.org/citation.cfm?id=3045118.3045367.

Karpathy, Andrej (2015). *The Unreasonable Effectiveness of Recurrent Neural Networks*. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

Martinsson, Egil (2016). "WTTE-RNN : Weibull Time To Event Recurrent Neural Network". MA thesis. Chalmers University Of Technology.

Nielsen, Michael A. (2015). *Neural Networks and Deep Learning*. Determination Press.

NIST/SEMATECH (2003). "6.3.1. What are Control Charts?" In: U.S. Department of Commerce. URL: https://www.itl.nist.gov/div898/handbook/pmc/section3/pmc31.htm.

Olah, Christopher. *Understanding LSTM Networks*. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Reid, Nancy (1994). "A Conversation with Sir David Cox". In: *Statistical Science*. Vol. 9.

Saxena, A. et al. (2008). "Damage propagation modeling for aircraft engine run-to-failure simulation". In: *2008 International Conference on Prognostics and Health Management*, pp. 1–9. DOI: 10.1109/PHM.2008.4711414.

Saxena, Abhinav and Kai Goebel (2008). "Turbofan Engine Degradation Simulation Data Set". In: URL: http://ti.arc.nasa.gov/project/prognostic-data-repository.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: 15, pp. 1929–1958.

XIA, Guo-en and Wei-dong JIN (2008). "Model of Customer Churn Prediction on Support Vector Machine". In: *Systems Engineering - Theory Practice*. Vol. 28, pp. 71–77.

Zaremba, Wojciech, Ilya Sutskever, and Oriol Vinyals (2014). "Recurrent Neural Network Regularization". In: *CoRR* abs/1409.2329.